Memory-Aware Embedded Control Systems Design

Wanli Chang, Dip Goswami, Samarjit Chakraborty, Lei Ju, Chun Jason Xue, and Sidharta Andalam

***Abstract*—Control applications are often implemented on highly cost-sensitive and resource-constrained embedded platforms, such as microcontrollers with a small on-chip memory. Typically, control algorithms are designed using model-based approaches, where the details of the implementation platform are completely ignored. As a result, optimizations that integrate platform-level characteristics into the control algorithms design are largely missing. With the emergence of cyber-physical systems (CPS)-oriented thinking, there has lately been a strong interest in co-design of control algorithms and their implementation platforms, leading to work on networked control systems and computation-aware control algorithms design. However, there has so far been no work on integrating the characteristics of a memory architecture into the design of control algorithms. In this paper we, for the first time, show that accounting for the impact of on-chip memory (or cache) reuse on the performance of control applications motivates new techniques for control algorithms design. This leads to significant improvement in quality of control for given resource availability, or more efficient implementations of embedded control applications. We believe that this paper opens up a variety of possibilities for memory-related optimizations of embedded control systems, that will be pursued by researchers working on computer-aided design for CPS.**

***Index Terms*—Embedded control systems, memory analysis, nonuniform sampling, quality of control (QoC).**

I. Introduction

TRADITIONALLY, control algorithms *design* and their *implementation* were strictly separated, the former being pursued by control theorists and the latter by embedded systems engineers. As a result, characteristics of the implementation platform related to—computation, communication, and memory hierarchy—are typically not accounted for in the control algorithms design process. With the emergence of the cyber-physical systems (CPS)-oriented thinking, there is now a strong interest in the co-design of control algorithms and their implementation platforms. Toward this, there has been a lot of recent work on control algorithms design that takes into account the characteristics of communication channels in networked embedded control systems, or schedule control tasks depending on real-time system states and disturbances to improve the quality of control (QoC) [1]–[3]. However, there has so far been no work that considers the characteristics of the memory architecture in the implementation platform when designing control algorithms.

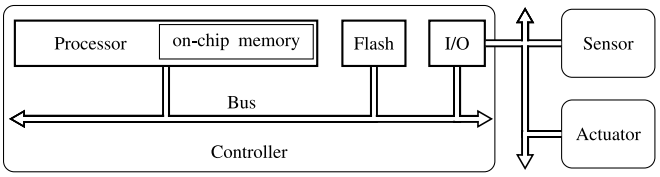


Fig. 1. Embedded control system with a processor and on-chip memory for program execution. Instructions are stored in the flash memory. Programmable I/Os are used for communication among the controller, sensors, and actuators.

In many cost-sensitive and resource-constrained embedded platforms, that are used to implement control algorithms, memory subsystems constitute an important component, and on-chip memories (used as caches) contribute significantly toward their cost. There have been many efforts on cache reuse maximization, for improving the real-time performance, such as the worst-case execution time (WCET), of embedded software [4]. However, the characteristics of control systems and metrics of control performance like QoC have not been directly incorporated into these techniques. In this paper, we follow a CPS-oriented approach and show that accounting for on-chip memory behavior motivates new techniques for control algorithms design, that are otherwise not used. These techniques open up a number of possibilities for co-design and co-optimization of control algorithms, code placement, and memory-aware control tasks scheduling, that we believe this paper will motivate other researchers to pursue.

The setup we consider is fairly general and occurs in several application domains, one example of which is automotive embedded systems. As illustrated in Fig. 1, the controller (such as the XC23xxB Series microcontroller [5] from Infineon that is popular in automotive systems) consists of an embedded processor with an on-chip memory (referred to as cache from here on) to run multiple control applications. The codes of these applications are stored in an external memory (often a flash memory), which has a large size and can thus accommodate all the application codes, but has high read/write latencies (hundreds of processor cycles). Programmable I/O peripherals are used for communication among the controller, sensors, and actuators.

In such an embedded implementation platform, the overall control loop for each application performs three operations.

1) Measuring the system states with sensors (*measure*).

Manuscript received June 16, 2015; revised October 30, 2015, February 25, 2016, and August 24, 2016; accepted September 4, 2016. Date of publication September 27, 2016; date of current version March 17, 2017. This work was supported by the Singapore National Research Foundation through its Campus for Research Excellence and Technological Enterprise Program. This paper was recommended by Associate Editor X. S. Hu.

W. Chang is with the Cluster of Information Communication Technology, Singapore Institute of Technology, Singapore 138683, and also with TUM CREATE, Singapore 138602 (e-mail: wanli.chang@singaporetech.edu.sg).

D. Goswami is with the Eindhoven University of Technology, 5600 Eindhoven, The Netherlands.

S. Chakraborty is with TU Munich, 80333 Munich, Germany.

L. Ju is with the Department of Computer Science and Technology, Shandong University, Jinan 250014, China.

C. J. Xue is with the City University of Hong Kong, Hong Kong.

S. Andalam is with the University of Auckland, 1142 Auckland, New Zealand.

Digital Object Identifier 10.1109/TCAD.2016.2613933

2) Computing the control input by executing the control application (*compute*).
3) Applying the computed control input to the plant (*actuate*).

The sampling period is the time duration between two consecutive measure operations. In general, a shorter sampling period implies faster response from the controller and thus better QoC. Assuming that measure and actuate operations require negligible time, the sampling period is constrained by the WCET of the control application.

Given a collection of control applications (e.g., $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$), it is conventional to run the control loops of them in a round-robin fashion ($\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, ...). Since the codes for different control applications are different, the on-chip cache, in this process is frequently refreshed. This results in poor cache reuse and long WCET. In order to address this issue, we propose a new sampling order for the control applications, using which cache reuse is improved and the WCET of each application is reduced. In particular, we study a nonuniform sampling scheme, where the control loop of each application is consecutively run multiple times—in order to increase cache reuse, before moving on to the next application (e.g., $\mathcal{C}_1$, $\mathcal{C}_1$, $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_2$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_3$, $\mathcal{C}_3$, ...). We design controllers for both the conventional memory-oblivious and the proposed memory-aware sampling orders. We then show that all else (e.g., the processor's operating frequency) remaining identical, the nonuniform sampling scheme results in significant improvement in the QoC (20%–30%, which is significant in highly cost-sensitive domains like automotive systems).

A. Contributions

As mentioned earlier, the impact of the memory architecture on control algorithms design has not been studied until now. The main contribution of this paper is a novel design flow for memory-aware embedded control systems that brings two very disparate classes of techniques—cache modeling and program analysis on one hand, and controller design on the other hand—together and quantifies the resulting benefit. The idea is to shorten the sampling periods of the control applications with execution (i.e., sampling) orders that increase the cache reuse. Such cache-aware execution of control applications implies nonuniform sampling with a shorter average sampling period. Generally, the nonuniform sampling scheme is undesirable in control systems, since the controller design has to deal with switching instability, making it challenging to optimize QoC. In our method, the key is that the sampling order is a design parameter and known in the controller design phase. Exploiting the knowledge of the sampling order and a shorter average sampling period, we propose a controller design technique that improves the QoC.

In the existing memory-conscious algorithms design (e.g., in real-time tasks), the programs were treated as black boxes and their functionality was not considered. In this paper, we seek to address this and explicitly consider the properties of control algorithms. In particular, the optimal choice of the control algorithms parameters, such as the gain values, are dependent on their sampling periods and sensing-to-actuation delays, which in turn are determined by the WCETs they experience. In the case of other algorithms, their parameter values are not updated on the basis of the WCETs. Hence, while memory-conscious algorithms design for other domains stops at WCET minimization, whether and how simultaneously modifying the controller parameters in response to the reduced WCET leads to improved QoC, is an open question. This paper makes the first efforts to answer this question.

While we exploit existing program analysis techniques in conjunction with cache modeling, our analysis focuses on estimating the guaranteed WCET reduction due to consecutive executions of the same program, which is required in computing the nonuniform sampling order for a feedback controller. Estimating such WCET reduction has not been studied before, mostly since until now there has been no useful context for studying it. In addition, QoC-optimal controller design with nonuniform sampling relies on our proposed technique that exploits the shortened WCET in the memory-aware sampling order. Hence, the contribution of this paper is the synergistically integrated framework of memory-aware embedded control systems design, which further provides new insights and design options in line with the CPS-oriented design philosophy, where the goal is to study control theory and embedded systems with the same footing.

B. Paper Organization

Section II presents the literature review on computation/communication-aware embedded control systems design and effective use of memory for embedded system performance improvement. Section III gives an overview of the proposed method for memory-aware embedded control systems design. In Section IV, the memory analysis technique is discussed and illustrated with a motivational example. Section V describes basics of feedback control applications under consideration and explains how the controller can be designed for the conventional memory-oblivious uniform sampling order and the proposed memory-aware nonuniform sampling scheme. Experimental results are shown in Sections VI and VII makes concluding remarks of this paper.

II. Related Work

Recent research work takes into account the implementation details, such as computation and communication, of embedded control systems in order to improve QoC and reduce the discrepancy between the design and implementation phases. In [1], a novel controller design technique based on a hierarchy of controllers is proposed, so that when the allocated execution time is short, a low-level computationally light controller is activated to achieve basic QoC and when the execution time is long, a high-level computationally intensive controller is used aiming for better QoC. The tradeoff between QoC and CPU usage is explored in [2] by dynamic scheduling of multiple self-triggered control tasks executed on one processor. The conventional paradigm in networked embedded control systems regards the messages as periodic, which facilitates the analysis and implementation, yet leads to conservative usage of the communication bandwidth. An aperiodic strategy for dynamic allocation of bandwidth according to the current state of the plants and available resources is proposed in [3]. None of the works have considered memory in the embedded control systems design, which is the focus of this paper.

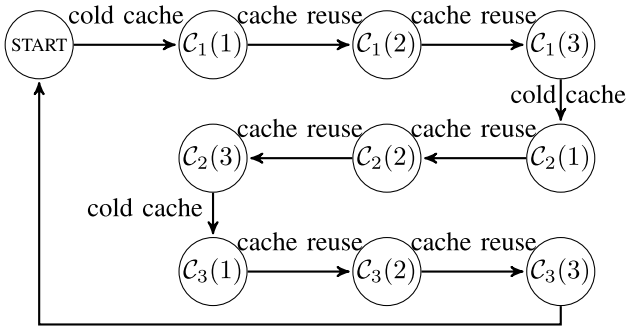


Fig. 2. Memory analysis of an example with three control applications. Each application is consecutively executed three times. After the first execution $\mathcal{C}_i(1)$, some instructions in the cache can be reused and thus the WCETs of the following two executions are shortened.

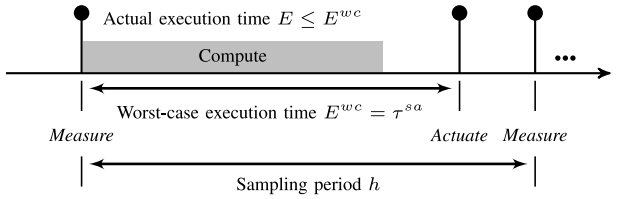


Fig. 3. General timing model of a control loop.

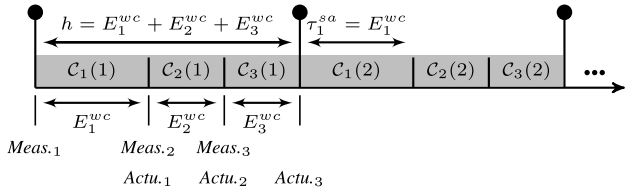


Fig. 4. In S1, there is no cache reuse. The WCET of all executions for the same application E_i^{wc} remains constant. The sampling period of every control application h is *uniform* under this scheme. The sensing-to-actuation delay τ_i^{sa} is equal to E_i^{wc}.

The tradeoff between memory and CPU usage in a feedback scheduling system, which dynamically adjusts the sampling periods of control tasks to maximize the overall QoC, is explored in [6]. However, it is not considered how memory can be exploited to improve the system performance. In [4], the round-robin scheduling of multiple tasks on an embedded operating system is dynamically tuned during program execution, adapting to changes in work load and external input stimulus. As a result, cache misses are reduced and the system performance is improved. Cache analysis for timing-related computation has been studied [7]–[10]. The architectural influence on static timing analysis of embedded hard real-time systems is described in [7]. The cache-related preemption delay is analyzed in [8] for a multitask embedded system with preemption, and extended to streaming applications in [9] for cache-aware timing estimation. The set-associative cache is considered in [10]. In this paper, we modify the existing approach to compute guaranteed WCET reduction due to cache reuse between two consecutive executions of the same application, which is then exploited by the tailored controller design method in the memory-aware sampling order to achieve better QoC.

Works in control theory literature with nonuniform sampling focus on guaranteeing stability of the resulting switched system [11]. Generally, theoretical tools such as common quadratic Lyapunov functions and switched Lyapunov functions tackle arbitrary switching between sampling periods to assure stability of the overall closed-loop system. In this paper, as opposed to arbitrary switching, the switching order is precisely known in the design phase, i.e., from the memory-aware sampling order. We aim for further performance optimality by exploiting this additional knowledge about the switching behavior. In the field of optimal control, techniques such as linear quadratic regulator [12] are well-developed. By adjusting the weights in the quadratic cost, a tradeoff between the input magnitude and the settling time can be achieved. However, these existing optimal control methods cannot be directly applied in this paper, since first, they are not specifically tuned for switched systems, and second, they do not explicitly consider the constraint on the input signal, which exists in all real-life systems. The combination of performance optimization and input constraints is addressed by model predictive control (MPC) techniques [13]—another well-developed area. First, MPC performs online optimization in every sampling period, making it computationally heavy and unsuitable for being implemented on the resource-constrained embedded platform, which is what we are studying in this paper. Second, the control law in MPC techniques is nonlinear in nature due to online optimization in every sample and existing literature on MPC does not explicitly handle switching between multiple linear sub-systems. Building upon these previous works discussed above, our method formulates an optimal pole-placement problem, where poles of sub-systems are decision variables, the input constraint is explicitly respected and the settling time is the optimization objective. Unlike MPC, our controller design is performed off-line making scalability a less important aspect.

III. Overview of the Proposed Method for Memory-Aware Embedded Control Systems Design

In the proposed memory-aware sampling order, each control application $\mathcal{C}_i$ is consecutively executed multiple times. An example with three applications ($\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$) is shown in Fig. 2, where $\mathcal{C}_i(j)$ denotes the jth execution of the control application $\mathcal{C}_i$. Before the first execution $\mathcal{C}_i(1)$, the cache is either empty (i.e., cold cache) or filled with instructions from other applications, that are not used by $\mathcal{C}_i$ (equivalent to cold cache). The WCET of $\mathcal{C}_i(1)$ can be computed by a number of existing standard techniques [7], [14], [15]. Before the second execution $\mathcal{C}_i(2)$, the instructions in the cache are from the same application $\mathcal{C}_i$ and thus can be reused. This results in more cache hits and hence shorter WCET. Depending on which path the program takes, the amount of WCET reduction varies. This requires a technique to compute the guaranteed WCET reduction of $\mathcal{C}_i(2)$ and $\mathcal{C}_i(3)$, independent of the path taken, which is presented in Section IV. To summarize, the WCET reduction results from the cache reuse increase, which is enabled by the change of the actual execution order of the applications, compared with the conventional round-robin fashion.

After WCET results are computed, the next task is to derive the control timing parameters (e.g., sampling periods and sensing-to-actuation delays). The general timing model of

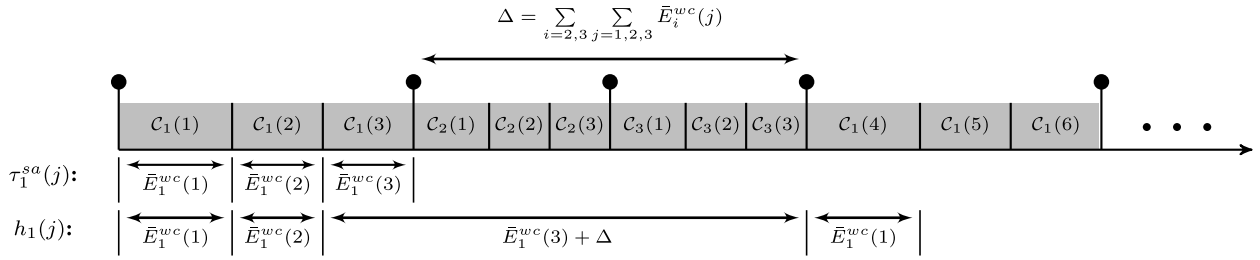


Fig. 5. In S2, the WCETs of the same control application vary, due to cache reuse. The sampling period for a control application is nonuniform.

a control loop is illustrated in Fig. 3. The compute operation executes the control application, which takes E time units. The sampling period is denoted by h. The time interval between the measure and the corresponding actuate operations in the same sampling period is sensing-to-actuation delay τ^{sa}, which is equal to the WCET of the control application E^{wc}.

In particular, we explore the relationship between WCET results and control parameters of two example sampling schemes. As illustrated in Fig. 4, S1 is the conventional memory-oblivious scheme and summarized as follows:

S1: $\mathcal{C}_1(1) \to \mathcal{C}_2(1) \to \mathcal{C}_3(1) \to \mathcal{C}_1(2) \to \mathcal{C}_2(2) \to \mathcal{C}_3(2) \to \mathcal{C}_1(3) \to \mathcal{C}_2(3) \to \mathcal{C}_3(3) \to \cdots$

There is no *cache reuse* in S1 as discussed above, considering that different control applications typically have different instructions to execute. In other words, when $\mathcal{C}_i(j)$ starts execution, all instructions of $\mathcal{C}_i$ need to be brought into the cache from the flash memory. Therefore

$$E_i^{\text{wc}}(1) = E_i^{\text{wc}}(2) = \cdots = E_i^{\text{wc}} \tag{1}$$

where $E_i^{\text{wc}}(j)$ is the WCET of the jth execution for $\mathcal{C}_i$. The WCET of the application $\mathcal{C}_i$ is denoted by E_i^{wc}, since all executions of the same application have equal WCET. Clearly, all control applications run with a uniform sampling period of

$$h = \sum_{i=1,2,3} E_i^{\text{wc}}. \tag{2}$$

Moreover, for the sensing-to-actuation delay

$$\tau_i^{\text{sa}} = E_i^{\text{wc}}. \tag{3}$$

As has been shown in Fig. 2, S2 is an example memory-aware sampling order and summarized as

S2: $\mathcal{C}_1(1) \to \mathcal{C}_1(2) \to \mathcal{C}_1(3) \to \mathcal{C}_2(1) \to \mathcal{C}_2(2) \to \mathcal{C}_2(3) \to \mathcal{C}_3(1) \to \mathcal{C}_3(2) \to \mathcal{C}_3(3) \to \cdots$

As illustrated in Fig. 5, we denote the effective WCET taking into account the cache reuse with $\bar{E}_i^{\text{wc}}(j)$. From the above discussion

$$\forall i \in \{1, 2, 3\}, \quad \bar{E}_i^{\text{wc}}(1) = E_i^{\text{wc}} \tag{4}$$

since there is no cache reuse for the first execution of every application $\mathcal{C}_i(1)$. $\bar{E}_i^{\text{wc}}(2)$ and $\bar{E}_i^{\text{wc}}(3)$ are shorter than $\bar{E}_i^{\text{wc}}(1)$ due to cache reuse. The amounts of cache reuse are the same for $\mathcal{C}_i(2)$ and $\mathcal{C}_i(3)$ in the worst case. Denoting the guaranteed WCET reduction as $\bar{E}_i^g$, we have

$\forall i \in \{1, 2, 3\}$

$$\bar{E}_i^{\text{wc}}(2) = \bar{E}_i^{\text{wc}}(3) = \bar{E}_i^{\text{wc}}(1) - \bar{E}_i^g. \tag{5}$$

From these varying WCETs, the sampling periods of all three applications can be calculated. Taking $\mathcal{C}_1$ as an example, there are three sampling periods $h_1(1)$, $h_1(2)$, and $h_1(3)$, which repeat themselves periodically

$$\begin{aligned} h_1(1) &= \bar{E}_1^{\text{wc}}(1), \quad h_1(2) = \bar{E}_1^{\text{wc}}(2) \\ h_1(3) &= \bar{E}_1^{\text{wc}}(3) + \Delta \end{aligned} \tag{6}$$

where Δ is computed as

$$\Delta = \sum_{i=2,3} \sum_{j=1,2,3} \bar{E}_i^{\text{wc}}(j). \tag{7}$$

Similar derivation can be done for $\mathcal{C}_2$ and $\mathcal{C}_3$. The average sampling period of an application h_{avg} is

$$h_{\text{avg}} = \frac{\sum_{i=1,2,3} \sum_{j=1,2,3} \bar{E}_i^{\text{wc}}(j)}{3} < h. \tag{8}$$

According to (4) and (5)

$$h_{\text{avg}} < \frac{\sum_{i=1,2,3} 3 \times E_i^{\text{wc}}}{3}. \tag{9}$$

From (2), we get

$$h_{\text{avg}} < h. \tag{10}$$

Moreover, the corresponding sensing-to-actuation delay $\tau_i^{\text{sa}}(j)$ also varies with cache reuse as

$\forall i \in \{1, 2, 3\}$

$$\begin{aligned} \tau_i^{\text{sa}}(1) &= h_i(1) = \bar{E}_i^{\text{wc}}(1) \\ \tau_i^{\text{sa}}(2) &= h_i(2) = \bar{E}_i^{\text{wc}}(2) \\ \tau_i^{\text{sa}}(3) &= \bar{E}_i^{\text{wc}}(3). \end{aligned} \tag{11}$$

As all control parameters have been derived, we can see that the sampling period $h_i(j)$ of a control application is *nonuniform* for the memory-aware scheme. The *average* sampling period of S2 is shorter than the uniform sampling period of S1 as shown in (8), due to WCET reduction resulting from cache reuse. The sensing-to-actuation delay $\tau_i^{\text{sa}}(j)$ varies as shown in (11). The next task is developing a controller design method to exploit shortened nonuniform sampling periods and achieve better QoC, which is reported in Section V. For the sake of comparison, the conventional controller design method for the memory-oblivious uniform sampling scheme S1 is also presented.

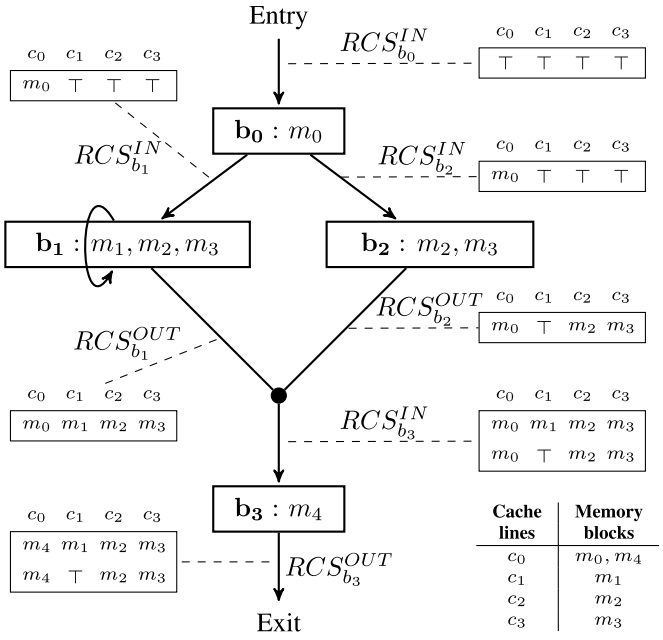


Fig. 6. Motivational example for memory analysis. Five memory blocks are mapped to four cache lines. Memory blocks executed by each basic block are shown. RCS^{IN} and RCS^{OUT} in the initialization phase are illustrated.

IV. Memory Analysis for Consecutive Executions of Control Application

In this section, the guaranteed WCET reduction for the second and subsequent executions of a control application in a memory-aware sampling scheme like S2. The technique is derived from previous research [8], [9] and modified to suit this paper. Similar to the data flow analysis, we start from a control flow graph (CFG) and then set up data flow equations for each node of the CFG [i.e., reaching cache state (RCS) and live cache state (LCS) computation in this context], based on which the fixed-point computation is performed. Afterward, the results of the fixed-point computation can be used to calculate the guaranteed WCET reduction. Fig. 6 presents a motivational example to illustrate our approach. Only instruction cache is considered in this paper.

A. Basics of Memory Analysis

In the typical automotive-use embedded control system shown in Fig. 1, on-chip memory works as cache and flash works as main memory. There are N_c cache lines, denoted as $\text{CL} = \{c_0, c_1, \ldots, c_{N_c-1}\}$ and the main memory has N_m blocks, denoted as $M = \{m_0, m_1, \ldots, m_{N_m-1}\}$. Each memory block is mapped to a fixed cache line. The example in Fig. 6 has four cache lines and five memory blocks. A basic block is a straight-line sequence of code with only one entry point and one exit point. This restriction makes a basic block highly amenable for program analysis. The presented CFG, consisting of four basic blocks $B = \{b_0, b_1, b_2, b_3\}$, has all the three key elements of a control program, i.e., sequential basic blocks, branches, and a loop. Therefore, it is suitable for illustrating our memory analysis technique.

There are three key terms in memory analysis that are described as follows.

1) *Cache States:* A cache state cs is described as a vector of N_c elements. Each element cs[i], where $i \in \{0, 1, \ldots, N_c - 1\}$, represents the memory block in the cache line c_i. When the cache line c_i holds the memory block m_j, where $j \in \{0, 1, \ldots, N_m - 1\}$, cs[$i$] $= m_j$. If c_i is empty, we denote it as cs[i] $= \perp$. If the memory block is unknown, we denote it as cs[i] $= \top$. CS is the set of all possible cache states.
2) *Reaching Cache States:* RCS of a basic block b_k, denoted as RCS_{b_k}, is the set of all possible cache states when b_k is reached via any incoming path.
3) *Live Cache States:* LCS of a basic block b_k, denoted as LCS_{b_k}, is the set of all possible first memory references to cache lines at b_k via any outgoing path.

Since our focus is on WCET reduction between two consecutive executions of $\mathcal{C}_i$, e.g., $\mathcal{C}_i(1)$ and $\mathcal{C}_i(2)$, we need to compute RCS of the exit point in $\mathcal{C}_i(1)$ and LCS of the entry point in $\mathcal{C}_i(2)$. By comparing all possible pairs of cache states, the guaranteed number of cache hits and thus WCET reduction can be calculated. In the following, we first discuss computation of RCS and LCS.

B. Computation of Cache States

In RCS computation, we first define gen_{b_k} as the cache state describing the last executed memory block in every cache line for the basic block b_k. Assuming that b_0 in Fig. 6 executes m_0 and then m_4, instead of only m_0, the last executed memory block in c_0 is m_4. Therefore, gen_{b_0} is $[m_4, \perp, \perp, \perp]$. For the example in Fig. 6

$$\begin{aligned} &\text{gen}_{b_0} = [m_0, \perp, \perp, \perp], \quad \text{gen}_{b_1} = [\perp, m_1, m_2, m_3] \\ &\text{gen}_{b_2} = [\perp, \perp, m_2, m_3], \quad \text{gen}_{b_3} = [m_4, \perp, \perp, \perp]. \end{aligned} \tag{12}$$

There are two equations involved in the RCS computation that calculate RCS^{IN} and RCS^{OUT}, where RCS^{IN} of a basic block b_k is the RCS before b_k is executed and RCS^{OUT} is the set of all possible cache states after b_k is executed. First, $\text{RCS}^{\text{OUT}}_{b_k}$ can be calculated from $\text{RCS}^{\text{IN}}_{b_k}$ as

$$\text{RCS}^{\text{OUT}}_{b_k} = \left\{ \mathcal{T}(b_k, \text{cs}) \middle| \text{cs} \in \text{RCS}^{\text{IN}}_{b_k} \right\} \tag{13}$$

where $\mathcal{T}$ is a transfer function defined as follows. For any cache state cs $\in$ CS and basic block $b_k \in B$, there is a cache state $\text{cs}' = \mathcal{T}(b_k, \text{cs})$, where for any cache line $c_i \in$ CL and $i \in \{0, 1, \ldots, N_c - 1\}$

$$\text{cs}'[i] = \begin{cases} \text{cs}[i]: & \text{if } \text{gen}_{b_k}[i] = \perp \\ \text{gen}_{b_k}[i]: & \text{otherwise.} \end{cases} \tag{14}$$

$\text{RCS}^{\text{IN}}_{b_k}$ can be calculated as

$$\text{RCS}^{\text{IN}}_{b_k} = \bigcup_{p \in \text{predecessor}(b_k)} \text{RCS}^{\text{OUT}}_{p} \tag{15}$$

where predecessor(b_k) is the set of all immediate predecessors of b_k.

The RCS computation is composed of two phases: initialization and fixed-point computation. As illustrated with the example in Fig. 6, the initialization phase starts from the entry basic block b_0 with $\text{RCS}^{\text{IN}}_{b_0} = \{[\top, \top, \top, \top]\}$. The element is $\top$ since our analysis is independent of the program executed before b_0. According to (13), $\text{RCS}^{\text{OUT}}_{b_0}$ is calculated to be $\{[m_0, \top, \top, \top]\}$. Since b_0 is the only immediate predecessor of b_2, $\text{RCS}^{\text{IN}}_{b_2}$ is equal to $\text{RCS}^{\text{OUT}}_{b_0}$ based on (15). Due to the

TABLE I
RCS COMPUTATION FOR THE MOTIVATIONAL EXAMPLE

	Basic Block	RCS^{IN}	RCS^{OUT}
Initialization	b_0	$\{[\top,\top,\top,\top]\}$	$\{[m_0,\top,\top,\top]\}$
	b_1	$\{[m_0,\top,\top,\top]\}$	$\{[m_0,m_1,m_2,m_3]\}$
	b_2	$\{[m_0,\top,\top,\top]\}$	$\{[m_0,\top,m_2,m_3]\}$
	b_3	$\{[m_0,m_1,m_2,m_3],[m_0,\top,m_2,m_3]\}$	$\{[m_4,m_1,m_2,m_3],[m_4,\top,m_2,m_3]\}$
Results from Fixed-Point Computation	b_0	$\{[\top,\top,\top,\top]\}$	$\{[m_0,\top,\top,\top]\}$
	b_1	$\{[m_0,\top,\top,\top],[m_0,m_1,m_2,m_3]\}$	$\{[m_0,m_1,m_2,m_3]\}$
	b_2	$\{[m_0,\top,\top,\top]\}$	$\{[m_0,\top,m_2,m_3]\}$
	b_3	$\{[m_0,m_1,m_2,m_3],[m_0,\top,m_2,m_3]\}$	$\{[m_4,m_1,m_2,m_3],[m_4,\top,m_2,m_3]\}$

TABLE II
LCS COMPUTATION FOR THE MOTIVATIONAL EXAMPLE

	Basic Block	LCS^{IN}	LCS^{OUT}
Initialization	b_3	$\{[\top,\top,\top,\top]\}$	$\{[m_4,\top,\top,\top]\}$
	b_2	$\{[m_4,\top,\top,\top]\}$	$\{[m_4,\top,m_2,m_3]\}$
	b_1	$\{[m_4,\top,\top,\top]\}$	$\{[m_4,m_1,m_2,m_3]\}$
	b_0	$\{[m_4,m_1,m_2,m_3],[m_4,\top,m_2,m_3]\}$	$\{[m_0,m_1,m_2,m_3],[m_0,\top,m_2,m_3]\}$
Results from Fixed-Point Computation	b_3	$\{[\top,\top,\top,\top]\}$	$\{[m_4,\top,\top,\top]\}$
	b_2	$\{[m_4,\top,\top,\top]\}$	$\{[m_4,\top,m_2,m_3]\}$
	b_1	$\{[m_4,\top,\top,\top],[m_4,m_1,m_2,m_3]\}$	$\{[m_4,m_1,m_2,m_3]\}$
	b_0	$\{[m_4,m_1,m_2,m_3],[m_4,\top,m_2,m_3]\}$	$\{[m_0,m_1,m_2,m_3],[m_0,\top,m_2,m_3]\}$

self loop, b_1 has both itself and b_0 as immediate predecessor. However, since $\text{RCS}_{b_1}^{\text{OUT}}$ has not been initialized yet, $\text{RCS}_{b_1}^{\text{IN}}$ is equal to $\text{RCS}_{b_0}^{\text{OUT}}$. In the same manner, we compute $\text{RCS}_{b_1}^{\text{OUT}}$, $\text{RCS}_{b_2}^{\text{OUT}}$, $\text{RCS}_{b_3}^{\text{IN}}$, and $\text{RCS}_{b_3}^{\text{OUT}}$, following the program flow as shown both in Fig. 6 and Table I. The initialization phase is completed once all basic blocks have been visited.

The next phase is fixed-point computation. RCS^{IN} and RCS^{OUT} of all basic blocks are computed iteratively with (13) and (15). This phase is terminated once the fixed point is reached, i.e., RCS^{IN} and RCS^{OUT} of all basic blocks remain unchanged. We let the program RCS be the RCS^{OUT} of the exit basic block, i.e., $\text{RCS} = \text{RCS}_{b_3}^{\text{OUT}}$. Results are reported in Table I.

The LCS computation can be done in a similar fashion. gen_{b_k} is defined as the cache state describing the first executed memory block in every cache line for the basic block b_k. Taking the same assumption when defining gen_{b_k} for RCS computation that b_0 in Fig. 6 executes m_0 and then m_4, instead of only m_0, the first executed memory block in c_0 is m_0. Therefore, gen_{b_0} is $[m_0, \perp, \perp, \perp]$. LCS^{IN} of a basic block b_k is the LCS after b_k is executed and can be derived from

$$\text{LCS}_{b_k}^{\text{IN}} = \bigcup_{s \in \text{successor}(b_k)} \text{LCS}_s^{\text{OUT}} \tag{16}$$

where successor(b_k) is the set of all immediate successors of b_k. LCS^{OUT} of b_k is the LCS before b_k is executed with

$$\text{LCS}_{b_k}^{\text{OUT}} = \left\{ \mathcal{T}(b_k, \text{cs}) \middle| \text{cs} \in \text{LCS}_{b_k}^{\text{IN}} \right\}. \tag{17}$$

LCS computation also comprises two phases of initialization and fixed-point computation. The only difference is that the initialization phase starts from the exit basic block and ends in the entry basic block. Detailed results for the motivational example are reported in Table II. We let the program LCS be the LCS^{OUT} of the entry basic block, i.e., $\text{LCS} = \text{LCS}_{b_0}^{\text{OUT}}$. It is noted that since the presented cache analysis technique is based on the fixed-point computation over the program CFG, it inherently handles loop structures in the CFG.

C. Guaranteed WCET Reduction

Conceptually, the program RCS is the set of all possible cache states after the program finishes execution by any execution path, and the program LCS is the set of all cache states, where each cache state contains memory blocks that may be first referenced after the program starts execution, for any execution path to follow. Both RCS and LCS could contain multiple cache states. Each pair with one cache state cs from the program RCS and one cache state cs′ from the program LCS represents one possible execution path between the two consecutive executions. For any cache line c_i in a pair, if cs[i] is equal to cs′[i] and they are not equal to $\top$, then there is certainly a hit and thus WCET reduction. Whether there is a hit for a particular cache line can be determined by the function $\mathcal{H}$ defined as follows.

$\forall \text{cs} \in \text{CS}$, $\text{cs}' \in \text{CS}$ and $c_i \in \text{CL}$, where $i \in \{0, 1, \ldots, N_c - 1\}$

$$\mathcal{H}(\text{cs}, \text{cs}', c_i) = \begin{cases} 1: & \text{if cs}[i] = \text{cs}'[i] \wedge \text{cs}[i] \neq \perp \\ 0: & \text{otherwise.} \end{cases} \tag{18}$$

The number of hits can be counted with the function $\mathcal{HT}$ defined as

$\forall \text{cs} \in \text{CS}$ and $\text{cs}' \in \text{CS}$

$$\mathcal{HT}(\text{cs}, \text{cs}') = \sum_{i=0}^{N_c - 1} \mathcal{H}(\text{cs}, \text{cs}', c_i). \tag{19}$$

The guaranteed number of hits among all possibilities is calculated as

$$\mathcal{G}(\text{RCS}, \text{LCS}) = \min_{\text{cs} \in \text{RCS}, \text{cs}' \in \text{LCS}} \left(\mathcal{HT}(\text{cs}, \text{cs}') \right). \tag{20}$$

Given that the main memory access time and the cache access time are, respectively, t_m and t_c, the guaranteed WCET reduction is computed as

$$\begin{aligned} \bar{E}^g &= \mathcal{G}(\text{RCS}, \text{LCS}) \times (t_m - t_c) \\ &\approx \mathcal{G}(\text{RCS}, \text{LCS}) \times t_m \end{aligned} \tag{21}$$

where the approximation can be taken if $t_c \ll t_m$.

For the motivational example, there are two cache states in RCS ($\text{RCS}_{b_3}^{\text{OUT}}$) and two cache states in LCS ($\text{LCS}_{b_0}^{\text{OUT}}$). In total, there are four pairs and the number of hits are calculated to be 3, 2, 2, and 2 with (19). For instance, $\mathcal{HT}([m_4, m_1, m_2, m_3], [m_0, m_1, m_2, m_3]) = 3$. Therefore, the guaranteed number of hits is 2 according to (20), no matter which path the program takes. From (21), the guaranteed WCET reduction is $2 \times (t_m - t_c)$, or approximately $2 \times t_m$, when $t_c \ll t_m$. It is noted that this result is obtained from the small example used for illustration. We expect more WCET reduction for larger realistic programs.

We remark that the direct-mapped cache (i.e., 1-way set-associative cache) is assumed in Fig. 6. The presented technique can be adapted to handle set-associative cache. For example, considering fully associative cache, when computing $\text{RCS}_{b_3}^{\text{OUT}}$ from $\text{RCS}_{b_3}^{\text{IN}}$, the memory block m_4 can be loaded to any cache line, which gives $\text{RCS}_{b_3}^{\text{OUT}}$ five more cache states, i.e., $[m_0, m_4, m_2, m_3]$, $[m_0, m_1, m_4, m_3]$, $[m_0, \top, m_4, m_3]$, $[m_0, m_1, m_2, m_4]$, and $[m_0, \top, m_2, m_4]$. From this, we can see that the number of cache states in RCS and LCS is larger for set-associative cache, which means that the guaranteed WCET reduction could be smaller. Details can be found in [10].

Using the memory analysis technique presented in this section, together with standard WCET analysis approaches, the effective WCET of $\mathcal{C}_i(2)$ and subsequent executions of $\mathcal{C}_i$ can be derived. Shorter WCET leads to smaller sampling period of the control system, which will be exploited in the next section to achieve better QoC.

V. Controller Design for Uniform and Nonuniform Sampling

In this section, we first describe the basics of feedback control applications under consideration. Then, it is explained how the controller can be designed for the memory-oblivious uniform sampling scheme. Lastly, the novel controller design approach tailored for nonuniform sampling is reported. The focus of this paper is on the linear state-feedback control of linear systems. Our proposed memory-aware embedded control systems design method can also be applied to nonlinear control law like MPC and nonlinear systems.

A. Basics of Feedback Control Applications

1) Plant Dynamics: A control application is responsible for controlling a plant or dynamic system. In particular, we consider linear single-input single-output (SISO) control applications, where the dynamic behavior is modeled by a set of differential equations

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \tag{22}$$

where $x(t) \in \mathbb{R}^n$ is the *system state*, $y(t)$ is the *system output* and $u(t)$ is the *control input*. The number of system states is n. There is often an overshoot constraint that $y(t)$ cannot be larger than a certain value. A, B, and C are *system matrices* of appropriate dimensions. These system matrices are physical properties of the plant. *System poles* are eigenvalues of A. In a state-feedback control algorithm, $u(t)$ is computed utilizing $x(t)$ (feedback signals) and then applied to the plant, which is expected to achieve certain desired behavior.

2) Discretized Dynamics: As discussed in Section III, in an embedded implementation platform, the overall control loop performs three operations: measure $x(t)$, compute $u(t)$, and apply $u(t)$. Generally, these operations are performed only at discrete time instants. The system is sampled at the measure operations. With the sampling period h, the continuous-time system in (22) can be transformed into a discrete-time system

$$\begin{aligned} x[k+1] &= A_d x[k] + B_d u[k] \\ y[k] &= C_d x[k] \end{aligned} \tag{23}$$

where sampling instants are $t = t_k$ $(k = 1, 2, 3, \ldots)$ and $h = t_{k+1} - t_k$. It is noted that h might not be a constant. $x[k]$ and $u[k]$ are the values of $x(t)$ and $u(t)$ at $t = t_k$ and

$$A_d = e^{Ah},\ B_d = \int_0^h \left(e^{At} dt\right) \cdot B,\ C_d = C. \tag{24}$$

Clearly, A_d and B_d are dependent on the sampling period h. We remark that the WCET of the control program is computed based on the instruction cache analysis and does not change with the exact model parameters of the system matrices, since the instructions and the cache/main memory access times remain unchanged.

3) System Controllability: Controllability of a discrete system is defined as the ability to transfer the system from any initial state $x[0] = x_0$ to any desired final state $x[k_f] = x_f$. The controllability matrix is

$$\mathcal{CO} = \begin{bmatrix} B_d & A_d B_d & A_d^2 B_d & \cdots & A_d^{n-1} B_d \end{bmatrix}. \tag{25}$$

A system is *controllable* if and only if $\mathcal{CO}$ is invertible, or equivalently, $\text{rank}(\mathcal{CO}) = n$.

4) Feedback Controller: In this paper, we consider regulation problems with constant output reference. That is, the overall control objective is to make $y[k] \to r$ as soon as possible, where r is the reference value. Toward this, we need to design $u[k]$ utilizing the states $x[k]$ in a state-feedback controller. The general structure is as follows:

$$u[k] = K \cdot x[k] + F \cdot r \tag{26}$$

where K is the feedback gain and F is the feedforward gain. In this state-feedback control, the data input of the control program mainly depends on the system state, which has n components, with each of them deriving its value from a certain range. Combining all components, the number of possible data inputs is very large, even after discretization.

5) Closed-Loop System: With the feedback controller as shown in (26), the system dynamics in (23) becomes

$$x[k+1] = (A_d + B_d K) x[k] + B_d F r \tag{27}$$

that is, closed-loop dynamics. Different locations of closed-loop system poles, i.e., eigenvalues of $(A_d + B_d K)$, result in different system behaviors. In *pole-placement*, the desired poles p can be decided with empirical or optimization techniques. This method is feasible since we have freedom to choose the feedback gain K. All eigenvalues of $(A_d + B_d K)$ must have absolute values of less than unity in order to ensure system stability [17].

6) Feedback and Feedforward Gain: Once pole locations are decided, we can construct the following characteristics equation of z with these poles as roots:

$$z^n + \gamma_1 z^{n-1} + \gamma_2 z^{n-2} + \cdots + \gamma_n = 0. \tag{28}$$

Then we define

$$\gamma_c(A_d) = A_d^n + \gamma_1 A_d^{n-1} + \gamma_2 A_d^{n-2} + \cdots + \gamma_n \mathbf{I} \tag{29}$$

where $\mathbf{I}$ is the n-dimensional identity matrix. According to Ackermann's formula [16], the feedback gain to stabilize the system is calculated as

$$K = [0 \quad \cdots \quad 0 \quad 1]\mathcal{CO}^{-1}\gamma_c(A_d). \tag{30}$$

The static feedforward gain F is designed to achieve $y[k] \to r$ and computed by

$$F = 1/\Big(C_d(\mathbf{I} - A_d - B_d K)^{-1} B_d\Big). \tag{31}$$

7) Restricted Pole-Placement: If the system is controllable, i.e., $\mathcal{CO}$ has full rank, there is no restriction on pole locations. The feedback gain K can be determined with (30). If $\mathcal{CO}$ does not have full rank, some of the poles cannot be modified with any choice of K and thus are *uncontrollable*. Since $\mathcal{CO}$ is not invertible, (30) does not work, either. In this case, if the uncontrollable poles are stable (with absolute values of less than unity), then the system is *stabilizable*. Restricted pole-placement can be used for stabilizable systems in the way that only controllable poles are placed in the desired locations and uncontrollable poles remain untouched. Therefore, for the embedded control systems design discussed in this paper, we require the system to be at least stabilizable, if not controllable.

8) QoC: There are various possible metrics to quantify QoC. In this paper, we consider *settling time* as our performance index. The time it takes for the system output $y[k]$ to reach and stay in a closed region around the reference value r (e.g., $0.98r$ to $1.02r$) is the settling time of a control loop. Shorter settling time implies better QoC.

9) Input Saturation: In almost every real-world system, there is some maximum available input signal and the controller needs to be designed such that the maximum value of $u[k]$ does not exceed this limit $U_{\max}$, i.e., $u[k] \le U_{\max}$. For example, in electric motor control, the magnitude of the input current is always limited.

B. Controller Design for Uniform Sampling

As can be derived from (2) and (3), for an application $\mathcal{C}_i$ under the conventional memory-oblivious sampling scheme S1, the constant sampling period h is larger than the constant sensing-to-actuation delay τ_i^{sa}. Therefore, the discrete-time system in (23) becomes a *sampled-data* system [18] as

$$x[k+1] = A_d x[k] + B_1\big(\tau_i^{\text{sa}}\big)u[k-1] + B_0\big(\tau_i^{\text{sa}}\big)u[k] \tag{32}$$

where

$$B_0\big(\tau_i^{\text{sa}}\big) = \int_0^{h-\tau_i^{\text{sa}}} e^{At}dt \cdot B, \;\; B_1\big(\tau_i^{\text{sa}}\big) = \int_{h-\tau_i^{\text{sa}}}^{h} e^{At}dt \cdot B.$$

In (32), it is assumed that $u[-1] = 0$ for $k = 0$. We notice that the system dynamics depends on both $u[k]$ and $u[k-1]$. Thus, we define a new system state $z[k] = [\,x[k] \quad u[k-1]\,]^T$ and the transformed system becomes

$$\begin{aligned} z[k+1] &= A_{S1}z[k] + B_{S1}u[k] \\ y[k] &= C_{S1}z[k] \end{aligned} \tag{33}$$

where

$$\begin{aligned} A_{S1} &= \begin{bmatrix} A_d & B_1\big(\tau_i^{\text{sa}}\big) \\ 0 & 0 \end{bmatrix} \\ B_{S1} &= \big[B_0\big(\tau_i^{\text{sa}}\big) \quad \mathbf{I}\big]^T, C_{S1} = \big[\,C_d \quad 0\,\big]. \end{aligned} \tag{34}$$

Next, we apply the following input signal:

$$u[k] = K_{S1} \cdot z[k] + F_{S1} \cdot r. \tag{35}$$

The closed-loop system is then

$$z[k+1] = (A_{S1} + B_{S1}K_{S1})z[k] + B_{S1}F_{S1}r. \tag{36}$$

In order to find the poles resulting in the best QoC with the pole-placement technique, we formulate a constrained optimization problem. Decision variables are the controllable closed-loop system poles, i.e., the controllable eigenvalues of $(A_{S1} + B_{S1}K_{S1})$. The optimization objective is the QoC. One constraint is that the closed-loop system is stable, i.e., the decision variables have absolute values of less than unity. The other constraint is the input saturation. This optimization problem can be solved by searching poles in the decision space and accelerated with heuristics, such as evolutionary algorithms. The feedback gain K_{S1} is then calculated according to (30). The feedforward gain F_{S1} is computed by (31). As long as (A_{S1}, B_{S1}) is *stabilizable*, i.e., uncontrollable poles have absolute values of less than unity, the above design is feasible. This design method is adapted from sampled-data systems literature and is suitable for the systems with known sensing-to-actuation delay shorter than the sampling period.

C. Controller Design for Nonuniform Sampling

As has been presented, in our proposed memory-aware embedded control systems design, the execution of the applications is reordered, such that the cache reuse can be increased. The increased cache reuse shortens the WCETs of the control programs and results in a nonuniform sampling order with a shorter average sampling period. In the following, we show how the controller is tailored to exploit these shortened WCETs to improve the QoC.

As discussed in Section III, the sampling period of a control application $\mathcal{C}_i$ in S2 varies as follows:

$$h_i(1) \to h_i(2) \to h_i(3) \to h_i(1) \to h_i(2) \to h_i(3) \to \cdots$$

As illustrated in Fig. 7, the dynamics switches periodically among the following three systems:

$$x[k+1] = A_d^1 x[k] + B_d^1 u[k] \tag{37}$$

$$x[k+2] = A_d^2 x[k+1] + B_d^2 u[k+1] \tag{38}$$

$$x[k+3] = A_d^3 x[k+2] + B_d^3 u[k+2]. \tag{39}$$

The output is then

$\forall l \in \{1, 2, 3\}$

$$y[k+l-1] = C_d^l x[k+l-1]. \tag{40}$$

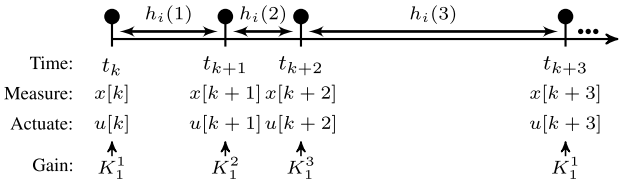


Fig. 7. Periodically switched sampling periods for C_i in S2.

Among them

$$A_d^l = e^{Ah_i(l)}, B_d^l = \int_0^{h_i(l)} \left(e^{At}dt\right) \cdot B, C_d^l = C. \tag{41}$$

Next, we need to design $u[k]$, $u[k+1]$, and $u[k+2]$ utilizing the available feedback signals.

1) Design of u[k]: The latest feedback signal available for computation of $u[k]$ is $x[k-1]$. Therefore, the input $u[k]$ is designed as follows:

$$u[k] = K_1^1 x[k-1] + F_1^1 r. \tag{42}$$

We define the new system state as

$$z[k] = \begin{bmatrix} x[k] \\ x[k-1] \end{bmatrix}. \tag{43}$$

The system in (37) is then transformed into the following:

$$\begin{aligned} z[k+1] &= A_1^1 z[k] + B_1^1 u[k] \\ y[k] &= C_1^1 z[k] \end{aligned} \tag{44}$$

where

$$\begin{aligned} A_1^1 &= \begin{bmatrix} A_d^1 & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \\ B_1^1 &= \begin{bmatrix} B_d^1 & \mathbf{0} \end{bmatrix}^T, C_1^1 = \begin{bmatrix} C_d^1 & \mathbf{0} \end{bmatrix}. \end{aligned} \tag{45}$$

The bold letter $\mathbf{0}$ denotes the zero matrix of an appropriate dimension. With the input in (42), the closed-loop dynamics of the system (37) in the sampling period $h_i(1)$ becomes

$$z[k+1] = A_{cl}^1 z[k] + B_1^1 F_1^1 r \tag{46}$$

where

$$A_{cl}^1 = \begin{bmatrix} A_d^1 & B_d^1 K_1^1 \\ \mathbf{I} & \mathbf{0} \end{bmatrix}. \tag{47}$$

The design of F_1^1 will be discussed later. As analyzed in Section V-A, the closed-loop system must be *stabilizable*, i.e., uncontrollable poles of A_{cl}^1 must have absolute values of less than unity. Decision variables are controllable poles of A_{cl}^1 and used to compute K_1^1. Details of the design method for K_1^1 with delayed input can be found in [19].

2) Design of u[k + 1]: With the augmented state shown in (43), the system dynamics in (38) becomes

$$\begin{aligned} z[k+2] &= A_1^2 z[k+1] + B_1^2 u[k+1] \\ y[k+1] &= C_1^2 z[k+1] \end{aligned} \tag{48}$$

where

$$\begin{aligned} A_1^2 &= \begin{bmatrix} A_d^2 & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \\ B_1^2 &= \begin{bmatrix} B_d^2 & \mathbf{0} \end{bmatrix}^T, C_1^2 = \begin{bmatrix} C_d^2 & \mathbf{0} \end{bmatrix}. \end{aligned} \tag{49}$$

By replacing $z[k+1]$ in (48) with (46), we obtain

$$z[k+2] = A_1^2 A_{cl}^1 z[k] + A_1^2 B_1^1 F_1^1 r + B_1^2 u[k+1]. \tag{50}$$

Since the latest feedback signal available for the computation of $u[k+1]$ is $x[k]$ as shown in Fig. 7, we use the following input signal in the sampling period $h_i(2)$:

$$u[k+1] = K_1^2 z[k] + F_1^2 r. \tag{51}$$

The overall closed-loop dynamics after the sampling period $h_1(2)$ becomes

$$z[k+2] = A_{cl}^2 z[k] + A_1^2 B_1^1 F_1^1 r + B_1^2 F_1^2 r \tag{52}$$

where

$$A_{cl}^2 = A_1^2 A_{cl}^1 + B_1^2 K_1^2. \tag{53}$$

Again the closed-loop system has to be *stabilizable*. Therefore, uncontrollable poles of A_{cl}^2 must have values of less than unity. Decision variables are controllable poles of A_{cl}^2 and used to compute K_1^2 according to (30). The design of F_1^2 will be discussed later.

3) Design of u[k + 2]: With the augmented state shown in (43), the system dynamics in (39) becomes

$$\begin{aligned} z[k+3] &= A_1^3 z[k+2] + B_1^3 u[k+2] \\ y[k+2] &= C_1^3 z[k+2] \end{aligned} \tag{54}$$

where

$$\begin{aligned} A_1^3 &= \begin{bmatrix} A_d^3 & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \\ B_1^3 &= \begin{bmatrix} B_d^3 & \mathbf{0} \end{bmatrix}^T, C_1^3 = \begin{bmatrix} C_d^3 & \mathbf{0} \end{bmatrix}. \end{aligned} \tag{55}$$

By replacing $z[k+2]$ in (54) with (52), we obtain

$$z[k+3] = A_1^3 A_{cl}^2 z[k] + B_1^3 u[k+2] + A_1^3 A_1^2 B_1^1 F_1^1 r + A_1^3 B_1^2 F_1^2 r. \tag{56}$$

For the above system, we use the following input signal in the sampling period $h_i(3)$:

$$u[k+2] = K_1^3 z[k] + F_1^3 r. \tag{57}$$

The overall closed-loop dynamics after the sampling period $h_1(3)$ becomes

$$z[k+3] = A_{cl}^3 z[k] + A_1^3 A_1^2 B_1^1 F_1^1 r + A_1^3 B_1^2 F_1^2 r + B_1^3 F_1^3 r \tag{58}$$

where

$$A_{cl}^3 = A_1^3 A_{cl}^2 + B_1^3 K_1^3. \tag{59}$$

In order for the closed-loop system to be *stabilizable*, uncontrollable poles of A_{cl}^3 must have values of less than unity. Decision variables are controllable poles of A_{cl}^3 and used to compute K_1^3 according to (30). To sum up, the pole-placement is possible and the feedback gains K_1^1, K_1^2, and K_1^3 can be designed using the presented technique as long as A_{cl}^1, A_{cl}^2, and A_{cl}^3 are *stabilizable*.

The pole-placement can be used to find the values of all decision variables, i.e., controllable poles of A_{cl}^1, A_{cl}^2, and A_{cl}^3, resulting in the optimal QoC while ensuring the stability and respecting the input saturation, either empirically or with optimization techniques. Feedback gains can then be computed as discussed before. In this paper, poles are empirically placed,

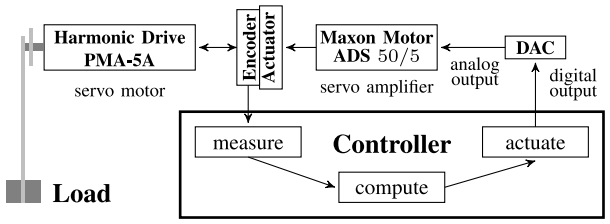


Fig. 8. Experimental setup of servo motor position control.

since our focus is to show the benefits of memory-aware embedded control systems design. It is possible to formulate the pole-placement optimization problem for nonuniform sampling, similar to the case of uniform sampling in Section V-B. However, unlike uniform sampling, the number of dimensions in the decision space increases linearly with the number of consecutive executions in this nonuniform sampling order. It is challenging to solve such a nonlinear and nonconvex optimization problem with large decision space, which can be part of future research.

4) Design of F_1^1, F_1^2, and F_1^3: The overall system dynamics for the entire cycle of three sampling periods $h_i(1)$, $h_i(2)$, and $h_i(3)$ can be obtained with (46), (52), and (58)

$$\begin{bmatrix} z[k+1] \\ z[k+2] \\ z[k+3] \end{bmatrix} = A_{cl} \cdot \begin{bmatrix} z[k] \\ z[k] \\ z[k] \end{bmatrix} + B_{cl} \cdot F_{S2} \cdot r$$
$$\begin{bmatrix} y[k] \\ y[k+1] \\ y[k+2] \end{bmatrix} = C_{cl} \cdot \begin{bmatrix} z[k] \\ z[k+1] \\ z[k+2] \end{bmatrix} \tag{60}$$

where

$$A_{cl} = \begin{bmatrix} A_{cl}^1 & \mathbf{0} & \mathbf{0} \\ A_{cl}^2 & \mathbf{0} & \mathbf{0} \\ A_{cl}^3 & \mathbf{0} & \mathbf{0} \end{bmatrix}$$
$$B_{cl} = \begin{bmatrix} B_1^1 & \mathbf{0} & \mathbf{0} \\ A_1^2 B_1^1 & B_1^2 & \mathbf{0} \\ A_1^3 A_1^2 B_1^1 & A_1^3 B_1^2 & B_1^3 \end{bmatrix}$$
$$C_{cl} = \begin{bmatrix} C_1^1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & C_1^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & C_1^3 \end{bmatrix}$$
$$F_{S2} = \begin{bmatrix} F_1^1 & F_1^2 & F_1^3 \end{bmatrix}^T. \tag{61}$$

Similar to the computation of the feedforward gain in (31), F_{S2} can be calculated as follows:

$$F_{S2} = \left(C_{cl}(\mathbf{I} - A_{cl})^{-1} B_{cl} \right)^{-1} \times \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T. \tag{62}$$

This controller design method is illustrated with the control application $\mathcal{C}_1$ under the sampling order S2 and can be applied for all stabilizable linear SISO control applications and memory-aware nonuniform sampling schemes.

As can be seen above, these gain values are determined in the controller design involving sampling periods and sensing-to-actuation delays, which are constrained by the WCETs. The cache state contains the instructions to be executed and changes as the program runs. Hence, the WCETs of the control programs are related to the cache states. The control system state changes depending on the control input, which is computed by the control program, based on the data input, gain values, and the instructions that are executed in the cache state. Therefore, there is an indirect link that relates the cache states to the system states.

TABLE III
EXPERIMENTAL CONFIGURATION FOR MEMORY ANALYSIS

Clock Frequency	Cache Lines	Cache Line Size	Hit/Miss Penalty
20 MHz	128	16 Byte	1/100 cycle

VI. EXPERIMENTAL RESULTS

In this paper, we consider a typical automotive-use controller, equipped with a processor, on-chip memory as cache and flash as main memory, shown in Fig. 1.[1] As a case study, we work on three control applications: $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. $\mathcal{C}_1$ is position control of a servo motor that can be used, e.g., in a steer-by-wire system [20]. The experimental setup is shown in Fig. 8. $\mathcal{C}_2$ is speed control of a dc motor that can be used in electric vehicle cruise control [21]. $\mathcal{C}_3$ is control of the electronic wedge brake system developed by Siemens as a brake-by-wire solution [22]. All three control applications run on the same processor.

In this section, for the two sampling orders S1 and S2 as described in Section III, first, the experimental configuration and time stamps based on the analysis shown in Section IV are presented. Then, the memory-oblivious and memory-aware controller designs are discussed, with the servo motor position control $\mathcal{C}_1$ as an example. Lastly, comparison in QoC of S1 and S2 for all three applications is reported.

A. Memory Analysis Experimental Configuration and WCET Results

As shown in Table III, the processor clock frequency is 20MHz. The cache is set to have 128 cache lines and each cache line is 16 bytes. When there is a cache hit, it takes 1 clock cycle to fetch the instruction and when there is a cache miss, it takes 100 clock cycles. Based on standard WCET analysis techniques applied to the control programs, the WCETs of all three applications without any cache reuse in S1 are computed to be

$$E_1^{\text{wc}} = 907.55\ \mu\text{s}, E_2^{\text{wc}} = 645.25\ \mu\text{s}, E_3^{\text{wc}} = 749.15\ \mu\text{s}.$$

Thus, the uniform sampling period as in (2) is

$$h = \sum_{i=1,2,3} E_i^{\text{wc}} = 2301.95\ \mu\text{s}.$$

The constant sensing-to-actuation delay τ_i^{sa} is given by (3) $\forall i \in \{1, 2, 3\}$

$$\tau_i^{\text{sa}} = E_i^{\text{wc}}.$$

In S2, according to (4) $\forall i \in \{1, 2, 3\}$

$$\bar{E}_i^{\text{wc}}(1) = E_i^{\text{wc}}.$$

[1]For instance, Infineon XC23xxB Series has a single processor with a minimum operating frequency of 20 MHz. It is typically equipped with a small size of on-chip SRAM memory and up to 256 kB flash memory.

TABLE IV
WCET RESULTS WITH AND WITHOUT CACHE REUSE FOR ALL THREE CONTROL APPLICATIONS

Application	WCET without Cache Reuse	Guaranteed WCET Reduction	WCET with Cache Reuse	WCET Reduction Percentage
$\mathcal{C}_1$	907.55 μs	455.40 μs	452.15 μs	50.18%
$\mathcal{C}_2$	645.25 μs	470.25 μs	175.00 μs	72.88%
$\mathcal{C}_3$	749.15 μs	514.80 μs	234.35 μs	68.72%

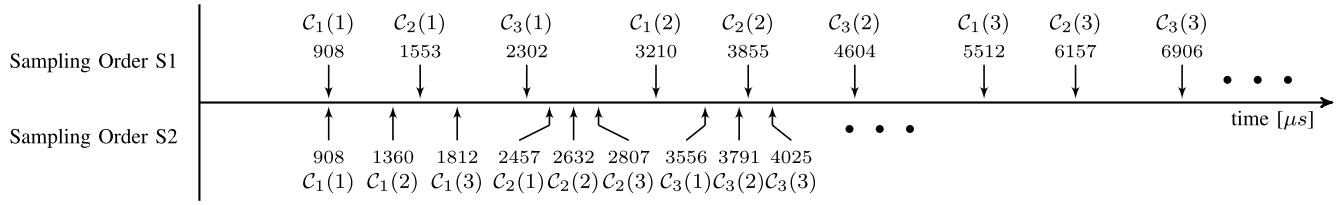


Fig. 9. Time stamps of both sampling orders S1 and S2 based on WCET results.

Based on the memory analysis approach presented in Section IV, the guaranteed numbers of cache hits for the three applications are

$$\mathcal{G}_1 = 92, \quad \mathcal{G}_2 = 95, \quad \mathcal{G}_3 = 104.$$

From the memory configuration in Table III, the cache access time is 1 processor clock cycle, i.e., $t_c = 0.05$ μs, and the main memory access time is 100 processor clock cycles, i.e., $t_m = 5$ μs. Therefore, according to (21), the guaranteed WCET reduction due to cache reuse for $\mathcal{C}_1$ can be calculated as

$$\bar{E}_1^g = \mathcal{G}_1 \times (t_m - t_c) = 92 \times (5 - 0.05)\ \mu\text{s} = 455.4\ \mu\text{s}.$$

Similarly, we have

$$\bar{E}_2^g = 470.25\ \mu\text{s}, \quad \bar{E}_3^g = 514.8\ \mu\text{s}.$$

According to (5), we can compute the reduced effective WCETs to be

$\forall j \in \{2, 3\}$

$$\bar{E}_1^{\text{wc}}(j) = 452.15\ \mu\text{s}, \bar{E}_2^{\text{wc}}(j) = 175\ \mu\text{s}, \bar{E}_3^{\text{wc}}(j) = 234.35\ \mu\text{s}.$$

Then the sampling periods can be obtained. Taking $\mathcal{C}_1$ as an example, using (6) and (7)

$$h_1(1) = 907.55\ \mu\text{s}, h_1(2) = 452.15\ \mu\text{s}, h_1(3) = 2665.25\ \mu\text{s}.$$

As in (8), the average sampling period of all three applications in S2 is calculated to be 1341.65 μs with 42% of reduction compared to the uniform sampling period in S1. The corresponding sensing-to-actuation delay $\tau_i^{\text{sa}}(j)$ is obtained with (11). WCET results with and without cache reuse for all three applications are shown in Table IV. Time stamps of both S1 and S2 are presented in Fig. 9.

B. Control Applications

We evaluate QoC of all three control applications to show the effectiveness of our proposed method. Details of memory-oblivious and memory-aware controller design are presented with $\mathcal{C}_1$ as an example. Controllers of $\mathcal{C}_2$ and $\mathcal{C}_3$ are designed with the same method. For $\mathcal{C}_1$, as shown in Fig. 8, the shaft of the servo motor (Harmonic Drive) [23] is attached to a rigid stick with 300g of weight at the end. The position of the motor shaft is measured by digital quadrature encoders attached to the motor shaft. The motor provides a desired amount of torque (computed by the control program) using a digital-to-analog converter (DAC) via a servo amplifier (Maxon Motor) [24].

1) Control Objective: The above system of servo motor position control has two states: $x_1(t)$, the angular position and $x_2(t)$, the angular velocity of the shaft. Initially, they are both set to be 0. The *measure* operation reads the quadrature encoder to obtain $x_1(t)$ and $x_2(t)$; the *compute* operation executes the control program and computes the input current $u(t)$ for the motor and the *actuate* operation is performed using the DAC. The control objective is to keep the rigid load at the angular position of 0.3 radian, i.e., $r = 0.3$rad. Since the *output* is the angular position $x_1(t)$, we have $y(t) = x_1(t)$. The dynamics of the above system is represented as in (22) with

$$A = \begin{bmatrix} 0 & 1 \\ 37 & 7.5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 6450 \end{bmatrix}, \quad C = [\,1 \quad 0\,].$$

For $\mathcal{C}_2$, the control objective is for the dc motor to reach a certain speed. For $\mathcal{C}_3$, the control objective is to achieve a certain braking force.

2) Input Saturation: In $\mathcal{C}_1$, the maximum current that the servo amplifier can supply to the motor is 1.5 ampere. Therefore, $|u(t)| \leq 1.5$ A must be respected by the controller. In $\mathcal{C}_2$, the input is the effective voltage of the battery pack that is used to power up the dc motor and cannot exceed 600 V. In $\mathcal{C}_3$, the control input is the power output to the brake wedge and the saturation is at 36 kW.

C. Controller Design for the Memory-Oblivious Uniform Sampling Scheme

As described in Section III, S1 is the conventional memory-oblivious sampling order. For $\mathcal{C}_1$, based on the timing results in Section VI-A, we obtain A_{S1} and B_{S1} as in (34)

$$A_{S1} = \begin{bmatrix} 1.0001 & 0.0023 & 0.0107 \\ 0.0844 & 0.9830 & 5.7764 \\ 0 & 0 & 0 \end{bmatrix}$$

$$B_{S1} = \begin{bmatrix} 0.0062 \\ 8.9446 \\ 1 \end{bmatrix}, \quad C_{S1} = [\,1 \quad 0 \quad 0\,].$$

By calculating the controllability matrix as in (25), (A_{S1}, B_{S1}) is controllable. Therefore, pole-placement is possible and we choose the pole locations by solving the optimization problem

$$p = [\,0.19 \quad 0.63 \quad 0.58\,].$$

TABLE V
FEEDBACK AND FEEDFORWARD GAINS FOR ALL THREE CONTROL APPLICATIONS

Application	S1		S2			
	feedback gain K_{S1}	feedforward gain F_{S1}	feedback gain K_1^1	feedback gain K_1^2	feedback gain K_1^3	feedforward gain F_{S2}
$\mathcal{C}_1$	$[-3.7212, -0.0432, -0.1734]$	3.7145	$[-4.8825, -0.049]$	$[-0.4174, 0.0651, -0.4174, -0.0855]$	$[-1.9086, -0.0617, -1.9086, 0.0198]$	$[4.8767, 0.8291, 3.8114]$
$\mathcal{C}_2$	$[-0.0193, -0.0043, -0.0916]$	0.02	$[-0.0067, -0.006]$	$[-0.0008, -0.0021, -0.0008, -0.0054]$	$[-0.0116, -0.0045, -0.0114, 0.0005]$	$[0.0075, 0.0024, 0.0237]$
$\mathcal{C}_3$	$[-6178.8, -81.1, -0.2]$	1.0816	$[22.1552, -80.327]$	$[-16884, -208, -16884, -12]$	$[-2491.8, -70.4, -2491.7, -13.7]$	$[0.2569, 4.485, 0.8832]$

With the above choice of poles, we obtain the controller gains as in (30) and (31)

$$K_{S1} = \begin{bmatrix} -3.7212 & -0.0432 & -0.1734 \end{bmatrix}, \quad F_{S1} = 3.7145.$$

Controllers of $\mathcal{C}_2$ and $\mathcal{C}_3$ are designed in the same way. Feedback and feedforward gains for all three applications under S1 are summarized in Table V.

D. Controller Design for Memory-Aware Nonuniform Sampling Order

We consider S2 described in Section III as an example memory-aware sampling order. The controller design of $\mathcal{C}_1$ is illustrated below.

1) Design of K_1^1: With delayed input in (42), the augmented system matrices in (45) are given by

$$A_1^1 = \begin{bmatrix} 1 & 0.0009 & 0 & 0 \\ 0.0335 & 0.9932 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$
$$B_1^1 = \begin{bmatrix} 0.0027 & 5.8367 & 0 & 0 \end{bmatrix}^T$$
$$C_1^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}.$$

The system (A_1^1, B_1^1) is stabilizable with a delayed input in (42). The closed-loop system poles are chosen to be

$$p_{1,1} = \begin{bmatrix} 0.86 & 0.51 & 0.6232 \end{bmatrix}$$

and

$$K_1^1 = \begin{bmatrix} -4.8825 & -0.049 \end{bmatrix}.$$

The closed-loop system matrix, as shown in (47), is given by

$$A_{cl}^1 = \begin{bmatrix} 1 & 0.0009 & -0.013 & -0.0001 \\ 0.0335 & 0.9932 & -28.4978 & -0.2863 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

2) Design of K_1^2: With input in (51), the augmented system matrices in (50) are given by

$$A_1^2 A_{cl}^1 = \begin{bmatrix} 1 & 0.0014 & -0.0258 & -0.0003 \\ 0.0501 & 0.9899 & -28.4016 & -0.2853 \\ 1 & 0.0009 & -0.013 & -0.0001 \\ 0.0335 & 0.9932 & -28.4978 & -0.2863 \end{bmatrix}$$
$$B_1^2 = \begin{bmatrix} 0.0007 & 2.9105 & 0 & 0 \end{bmatrix}^T$$
$$C_1^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}.$$

It can be verified that $(A_1^2 A_{cl}^1, B_1^2)$ is not controllable but stabilizable since the uncontrollable pole is located at 0. The controllable poles are placed at

$$p_{1,2} = \begin{bmatrix} 0.77 & 0.33 & 0.78 \end{bmatrix}.$$

The above location of poles leads to the following feedback gain:

$$K_1^2 = \begin{bmatrix} -0.4174 & 0.0651 & -0.4174 & -0.0855 \end{bmatrix}.$$

Next, the closed-loop system matrix A_{cl}^2 after the sampling period $h_1(2)$ can be computed as shown in (53)

$$A_{cl}^2 = \begin{bmatrix} 0.9998 & 0.0014 & -0.0261 & -0.0003 \\ -1.1649 & 1.1795 & -29.6166 & -0.5341 \\ 1 & 0.0009 & -0.013 & -0.0001 \\ 0.0335 & 0.9932 & -28.4978 & -0.2863 \end{bmatrix}.$$

3) Design of K_1^3: With input in (57), the augmented system matrices in (56) are given by

$$A_1^3 A_{cl}^2 = \begin{bmatrix} 0.9968 & 0.0045 & -0.1042 & -0.0017 \\ -1.0444 & 1.1564 & -29.0371 & -0.5236 \\ 0.9998 & 0.0014 & -0.0261 & -0.0003 \\ -1.1649 & 1.1795 & -29.6166 & -0.5341 \end{bmatrix}$$
$$B_1^3 = \begin{bmatrix} 0.0227 & 17.013 & 0 & 0 \end{bmatrix}^T$$
$$C_1^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}.$$

Here, $(A_1^3 A_{cl}^2, B_1^3)$ is not controllable but stabilizable since the uncontrollable pole is located at 0. The controllable poles are placed at

$$p_{1,3} = \begin{bmatrix} 0.12 & 0.08 & 0.3 \end{bmatrix}.$$

The above location of poles leads to the following feedback gain:

$$K_1^3 = \begin{bmatrix} -1.9086 & -0.0617 & -1.9086 & 0.0198 \end{bmatrix}.$$

Next, the closed-loop system matrix A_{cl}^3 after the sampling period $h_1(3)$ as shown in (59) is

$$A_{cl}^3 = \begin{bmatrix} 0.9534 & 0.0031 & -0.1476 & -0.0013 \\ -33.5156 & 0.1068 & -61.5073 & -0.1875 \\ 0.9998 & 0.0014 & -0.0261 & -0.0003 \\ -1.1649 & 1.1795 & -29.6166 & -0.5341 \end{bmatrix}.$$

The feedforward gain is computed by (62) as

$$F_{S2} = \begin{bmatrix} 4.8767 & 0.8291 & 3.8114 \end{bmatrix}^T.$$

Controllers of $\mathcal{C}_2$ and $\mathcal{C}_3$ are designed with the same method. Feedback and feedforward gains of all three applications under S2 are summarized in Table V. It can be seen that the gains of $\mathcal{C}_2$ are small and that the gains of $\mathcal{C}_3$ are large. This difference is mainly due to the fact that: 1) different control applications have different control input saturations and 2) the system state values of different control applications are in different ranges.

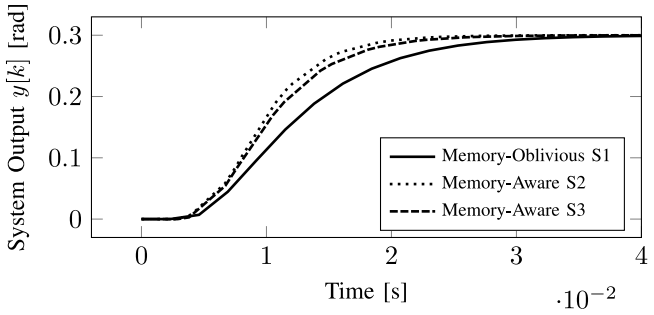


Fig. 10. Control system output of three different sampling orders for the control application $\mathcal{C}_1$.

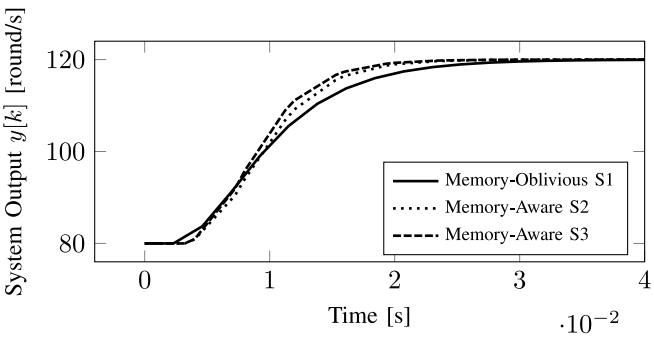


Fig. 11. Control system output of three different sampling orders for the control application $\mathcal{C}_2$.

E. QoC Improvement of All Control Applications

In order to show that S2 is not a singular sampling order that is able to achieve better QoC, besides S1 and S2 introduced in Section III, we evaluate the QoC of another memory-aware sampling order S3 in the experiment

S3: $\mathcal{C}_1(1) \rightarrow \mathcal{C}_1(2) \rightarrow \mathcal{C}_2(1) \rightarrow \mathcal{C}_2(2) \rightarrow \mathcal{C}_2(3) \rightarrow \mathcal{C}_3(1) \rightarrow \mathcal{C}_3(2) \rightarrow \mathcal{C}_3(3) \rightarrow \mathcal{C}_3(4) \rightarrow \cdots$

The control system outputs of all three sampling orders for all three applications are presented in Figs. 10–12. QoC comparison among S1, S2, and S3 for $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ is reported in Table VI. It can be seen that both memory-aware sampling orders improve the QoC compared to the conventional memory-oblivious sampling order in all control applications. By comparing S2 and S3, S2 results in better QoC in $\mathcal{C}_1$, but S3 achieves better QoC in $\mathcal{C}_2$ and $\mathcal{C}_3$. Neither of them dominates the other in all applications. Intuitively, the more frequent an application is sampled, the better performance could be achieved. For $\mathcal{C}_1$, the average sampling period in S2 is 1341.65 μs as computed in Section VI-A, 29.5% smaller than the average sampling period in S3 1903.58 μs, which can be computed from the WCET results reported in Table IV. Therefore, QoC of $\mathcal{C}_1$ in S2 is better than that in S3. Similarly, the average sampling period in S3 for $\mathcal{C}_3$ is 951.79 μs, 29.1% smaller than the average sampling period in S2. Therefore, QoC of $\mathcal{C}_3$ in S3 is better than that in S2. For $\mathcal{C}_2$, the average sampling period in S3 is 1269.05 μs, which is close to the average sampling period in S2. After the evaluation, we find out that QoC in S3 is better than QoC in S2. It is an interesting problem to find the optimal sampling order that maximizes the system QoC, which will be the future work. Considering the optimal sampling order, the QoC improvement

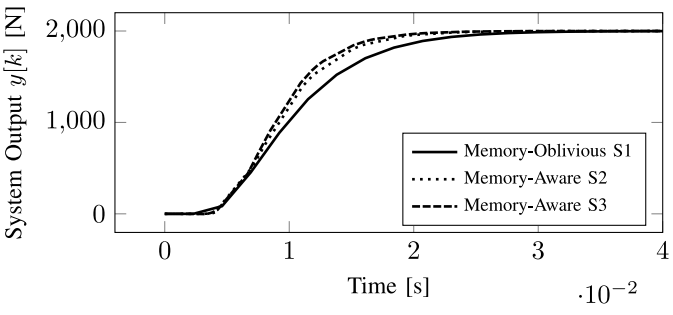


Fig. 12. Control system output of three different sampling orders for the control application $\mathcal{C}_3$.

TABLE VI
QoC Comparison for All Three Applications Among Three Sampling Orders

Application	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
Settling Time for S1 [ms]	31.2	26.8	25.2
Settling Time for S2 [ms]	21.5	21.1	20.4
Settling Time for S3 [ms]	23.9	20.8	19.4
QoC Improvement of S2 Compared to S1 [%]	31.1	21.3	19.0
QoC Improvement of S3 Compared to S1 [%]	23.4	22.4	23.0

from memory-aware embedded control systems design can be even higher than what is presented in this paper.

VII. Conclusion

This paper follows the line of work considering implementation platforms during the design of embedded control systems in order to improve QoC and reduce the deviation between the design and implementation phases. In particular, we investigate the impact of a memory-aware sampling order on control systems. While as the first paper considering memory in embedded control systems design, we specifically exploit the reuse of direct-mapped cache, the presented technique can be extended to handle other types of cache (e.g., set-associative cache) and memory (e.g., scratchpad). The proposed framework can be used to evaluate the impact of other memory-related problems, such as cache locking and scratchpad allocation, on control applications. The benefits of using a cache-aware controller will be even more for computationally more expensive control algorithms, such as MPC. However, the goal of this paper is to make a case for cache memory-aware controller design. How such memory-aware controllers may be designed for different classes of control algorithms, we believe, would be a topic of future papers also from other research groups.

From the control-theoretic perspective, the solution proposed in this paper is targeting regulation problems with constant output reference. As has been presented, we use static feedforward gain to regulate the control output. A time-varying reference (e.g., sinusoidal) requires a tracking control algorithm, which uses dynamic feedforward gain to reflect the variation in reference with time. A meaningful QoC for a tracking problem is the sum of error over time. Due to the nonuniform nature of sampling in this paper, further investigation is required for a suitable tracking algorithm, which is a part of future work.

After proving that the memory-aware sampling scheme does have considerable influence on control system performance,

the next problem to solve is finding the optimal sampling order that maximizes the system QoC. It consists of two stages. First, given a sampling order, we need to find the optimal poles that maximize the system QoC while the input saturation constraint is respected. This is a challenging problem to solve, due to the nonconvexity, nonlinearity and many-dimensional decision space as discussed before. Second, based on the results from the first stage, the optimal sampling order needs to be found. As the number of applications grows, the number of possible periodic sampling orders increases exponentially. Considering that the QoC evaluation of one sampling order is computationally heavy, brute force is practically infeasible. Heuristic methods need to be developed that are able to achieve a flexible balance between optimality and scalability. This will be a piece of future work along the research path of memory-aware embedded control systems design.

REFERENCES

[1] L. Greco, D. Fontanelli, and A. Bicchi, "Design and stability analysis for anytime control via stochastic scheduling," *IEEE Trans. Autom. Control*, vol. 56, no. 3, pp. 571–585, Mar. 2011.

[2] S. Samii, P. Eles, Z. Peng, P. Tabuada, and A. Cervin, "Dynamic scheduling and control-quality optimization of self-triggered control applications," in *Proc. RTSS*, San Diego, CA, USA, 2010, pp. 95–104.

[3] A. Anta and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over CAN," in *Proc. RTSS*, Washington, DC, USA, 2009, pp. 3–12.

[4] K. W. Batcher and R. A. Walker, "Dynamic round-robin task scheduling to reduce cache misses for embedded systems," in *Proc. DATE*, Munich, Germany, 2008, pp. 260–263.

[5] Infineon. (2009). *Product Brief*. [Online]. Available: http://www.infineon.com/dgdl/Pb_XC2300B.pdf?fileId=db3a30432a7fedfc012ab3c3d7863706

[6] S. G. Robertz, D. Henriksson, and A. Cervin, "Memory-aware feedback scheduling of control tasks," in *Proc. ETFA*, Prague, Czech Republic, 2006, pp. 70–77.

[7] R. Wilhelm *et al.*, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 7, pp. 966–978, Jul. 2009.

[8] H. S. Negi, T. Mitra, and A. Roychoudhury, "Accurate estimation of cache-related preemption delay," in *Proc. CODES*, 2003, pp. 201–206.

[9] S. Chakraborty, T. Mitra, A. Roychoudhury, and L. Thiele, "Cache-aware timing analysis of streaming applications," *Real Time Syst.*, vol. 41, no. 1, pp. 52–85, 2009.

[10] J. C. Kleinsorge, H. Falk, and P. Marwedel, "A synergetic approach to accurate analysis of cache-related preemption delay," in *Proc. EMSOFT*, Taipei, Taiwan, 2011, pp. 329–338.

[11] H. Lin and P. J. Antsaklis, "Stability and stabilizability of switched linear systems: A survey of recent results," *IEEE Trans. Autom. Control*, vol. 54, no. 2, pp. 308–322, Feb. 2009.

[12] E. Lavretsky and K. A. Wise, *Robust and Adaptive Control*. London, U.K.: Springer, 2013.

[13] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Madison, WI, USA: Nob Hill, 2009.

[14] R. Wilhelm *et al*., "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, p. 36, 2008.

[15] S. Andalam, A. Girault, R. Sinha, P. Roop, and J. Reineke, "Precise timing analysis for direct-mapped caches," in *Proc. DAC*, Austin, TX, USA, 2013, p. 148.

[16] J. Ackermann and V. Utkin, "Sliding mode control design based on Ackermann's formula," *IEEE Trans. Autom. Control*, vol. 43, no. 2, pp. 234–237, Feb. 1998.

[17] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton Univ. Press, 2009.

[18] A. Y. Bhave and B. H. Krogh, "Performance bounds on state-feedback controllers with network delay," in *Proc. CDC*, Cancún, Mexico, 2008, pp. 4608–4613.

[19] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 6, pp. 2337–2345, Nov. 2014.

[20] P. Yih, "Steer-by-wire: Implications for vehicle handling and safety," Ph.D. dissertation, Dept. Mech. Eng., Stanford Univ., Stanford, CA, USA, 2005.

[21] W. Chang, A. Pröbstl, D. Goswami, M. Zamani, and S. Chakraborty, "Battery-and aging-aware embedded control systems for electric vehicles," in *Proc. RTSS*, Rome, Italy, 2014, pp. 238–248.

[22] J. Fox *et al.*, "Modeling and control of a single motor electronic wedge brake," Siemens VDO Autom., Siemens AG, Regensburg, Germany, SAE Tech. Rep. 2007-01-0866, 2007.

[23] Harmonic Drive. (2016). *Produktbeschreibung PMA*. [Online]. Available: http://harmonicdrive.de/produkte/media/catalog/category/pma_catalogue_3.pdf

[24] Maxon Motor. (2011). *Product Specifications*. [Online]. Available: http://www.maxonmotor.com/medias/sys_master/root/8796918153246/ADS-145391-11-EN-281-282.pdf

Wanli Chang received the Ph.D. degree in electrical and computer engineering from the Technical University of Munich, Munich, Germany.

He is a Lecturer with the Singapore Institute of Technology, Singapore. His current research interest includes resource-aware automotive control systems.

Dip Goswami received the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2009.

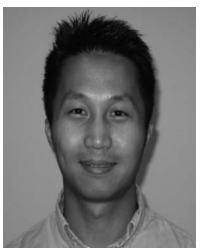
He is an Assistant Professor with the Eindhoven University of Technology, Eindhoven, The Netherlands. His current research interests include embedded control systems and cyber-physical systems and robotics.

Samarjit Chakraborty received the Ph.D. degree in electrical and computer engineering from ETH Zurich, Zürich, Switzerland, in 2003.

He is a Professor of Electrical and Computer Engineering with the Technical University of Munich, Munich, Germany. His current research interests include embedded and cyber-physical systems and software design.

Lei Ju received the Ph.D. degree from the National University of Singapore, Singapore.

He is an Associate Professor with the School of Computer Science and Technology, Shandong University, Jinan, China. His current research interests include memory architecture design and optimization and heterogeneous computing.



Chun Jason Xue received the Ph.D. degree from the University of Texas at Dallas, Richardson, TX, USA, in 2007.

He is an Associate Professor with the Computer Science Department, City University of Hong Kong, Hong Kong. His current research interests include nonvolatile memories, embedded, and real-time systems.

Sidharta Andalam received the Ph.D. degree from the University of Auckland, Auckland, New Zealand.

He is a Research Fellow with the University of Auckland. His current research interests include embedded systems, hybrid systems, automotive software, synchronous languages, and software platform for developing Internet of Things applications.